Vision Based Recognition of Vehicle Types

Study Thesis

Interactive Systems Laboratories
Prof. Dr. Alex Waibel

Faculty of Computer Science
University of Karlsruhe, Germany
Carnegie Mellon University of Pittsburgh, USA

by

Daniel Morlock

10/27/2008

Advisors:

Prof. Dr. Alex Waibel
Dr.-Ing. Rainer Stiefelhagen
Dr. Jie Yang
Dr. Yan Li

Herby I state that I created this paper all by myself and that no other resources than the stated ones were used.

Karlsruhe, 10/27/08 ....

Contents

1 Introduction

1.1 Motivation

The invention of the automobile changed our daily life significantly. After the car became affordable for the general public, the number of used cars increased dramatically. Today, the number of vehicles in the public traffic is valued at about 780 millions and is estimated to be more than 1 billion at the end of the year 2020. Thus, the car became a constant element in human life.
Of course, automobiles do not only bring along advantages. According to the world health organisation (WHO), about 1.2 million people are killed in a car accident per year. With the number of cars on the street, the number of deaths on the road is increasing, too. Even walking on the road has become more dangerous than never before.
Today, modern computer systems support people to handle the daily traffic. For drivers and walkers, it is necessary to detect and recognize other vehicles as early as possible in order to react appropriately. This is generally quiet easy for human beings. But the question is whether this can be done by computers, too?

1.2 Problem Formulation

The aim of this work is to recognize the type of a vehicle by using images or video sequences. By using sequential images, the separation of the vehicle from the background can be done by using differential images or Kalman filters [1]. But what is about non-moving vehicles? All objects which resist in their position become part of the background. In those cases, sequential images represent the same information as one single image and we can concentrate on vehicle type recognition based on a single image.
Yan Li and Takeo Kanade work on a car detection and registration system which can be used to find out the position and shape of each vehicle in a given input image. Further, the pre-processing system provides information about the vehicle orientation so that we can use methods for feature extraction depending on the current view of the vehicle. In this work, we extend the detection and registration system of Yan Li and Takeo Kanade with an automated recognition of the vehicle type. We deal with four different vehicle categories shown in figure 1.1. We

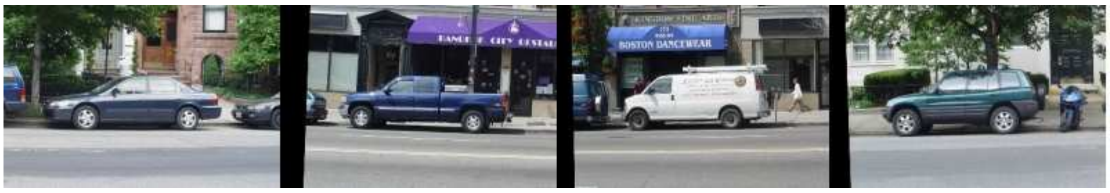


Figure 1.1: Supported vehicle types: Car, Pickup, Van and Jeep

distinguish between cars, pickups, vans and jeeps. The recognition extension is able to classify the vehicle type from five different viewpoints: Head, left head, side, left tail and tail. The right viewpoints are invariant against the left. Thus we can ignore the right head, right side and right tail viewpoint.

1.3 System Overview

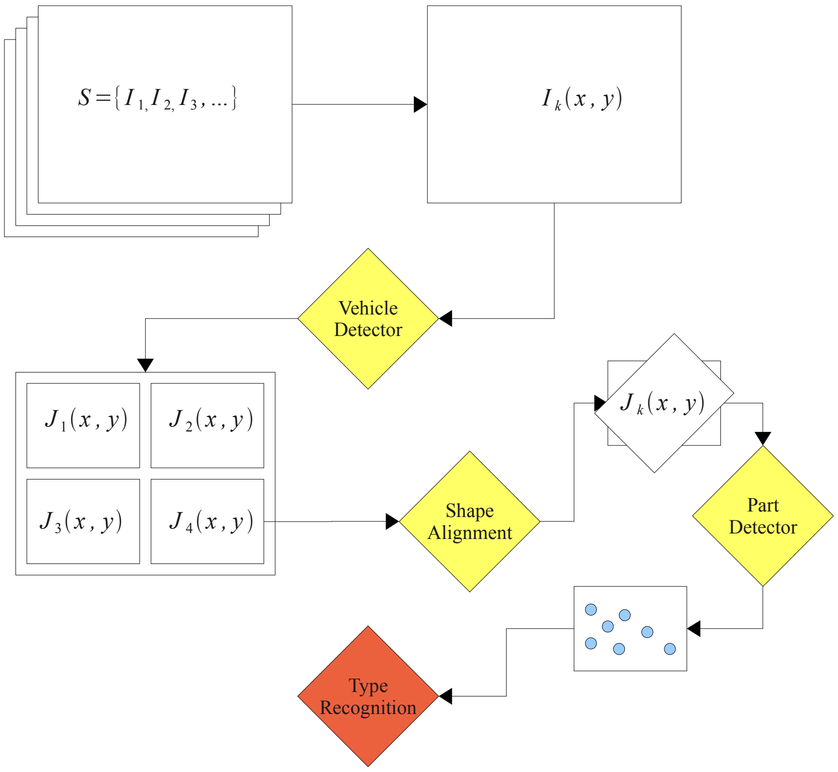


Figure 1.2: System Overview: Detection, Alignment and Recognition

Figure 1.2 shows the system architecture. As we can see, we expect a single image $I_k(x, y)$ as input image. This image can be part of any sequential images $S = \{I_1, I_2, \dots\}$ and it can contain more than only one vehicle. The vehicle detector tries to extract the vehicles by scanning the image with a 128×128 sized

sub-window. In each scan, an histogram of orientation gradients (HoG) is formed from the pixels within that sub-window. The detector uses multiclass logitboost to select discriminative features and returns a set of landmark points which describe the vehicle position. We can use this information to extract the car and create a new image $J_k(x, y)$ [2].
The extracted vehicle image and the set of landmark points form the parameter for the next system part. The alignment component uses one of the alignment methods described in chapter 2.2 and converts the vehicle images so that every vehicle has the same rotation and position as all other vehicles in the corresponding viewpoint class.
The part detector scans the image a second time and returns the position of characteristic vehicle parts like the windshield, the hood or the radiator grill. We will see that we can use this information for speedup and to improve the recognition accuracy.
The last part of the system uses the information of the previous components and tries to classify the vehicle type. The following work will describe several approaches and evaluate them in regard to recognition accuracy and performance.

1.4 Applications

Video surveillance plays an increasing role in public life. More and more highways, intersections or whole cities use video surveillance in order to regulate the huge traffic volume. In most cases, an ordinary traffic census system is used which counts passing vehicles by using infrared or simple computer vision approaches. Apart from the number of vehicles, it would be interesting to know the type of vehicle which passed the observation point. This information could greatly influence decisions made for the traffic system. Highways could be build according to the type of vehicles that will use them. New intersections or traffic lights can be installed which react intelligently to the currently dominating vehicle types.
But a vehicle detection and recognition system is not only useful for video surveillance. If we think about navigation systems, we could provide useful information for car drivers. For example, the navigation system can automatically keep a security distance which depends on the type of the vehicle in front of us.
Vehicle recognition also plays an important role in cognitive automobiles. If we want a computer to drive our car through city traffic, it has to know the position, size and behaviour of the cars around. Thus a motorcycle can be overtaken easily while we need more space if the want to overtake a minivan or a pickup truck.

1.5 Related Work

Peijin Ji, Lianwen Jin and Xutao Li used a partial Gabor filter bank for vision based vehicle type classification [3]. They extracted features from both edge images and grey images from the side view of the car by using a partial Gabor filter bank and different sampling methods. The dimension of the feature space was reduced by using Principal Component Analysis. A minimum distance classifier derived from the Bayes' decision theory was used to classify the vehicle into five different categories: Sedan, van, hatchback sedan, bus and van truck. They ran experiments with a testing database of total 1196 vehicle side views and achieved a maximum recognition accuracy of 95.17%.

Peter Shin, Hector Jasso, Sameer Tilat, Neil Cotofana and Tony Fountain from the Department of Structural Engineering at the University of California used strain histories from installed bridge-deck panels and tried to classify the passing vehicles using Naive Bayesian, a Neural Network and a Support Vector Machine at a time [4]. They collected strain histories from 2100 vehicles, normalized them and trained the different analytical methods by using about 400 vehicles from each of the five categories: Small vehicle, medium truck, bus, 3-axle truck and combination truck. In their experiments, they achieved a recognition accuracy of 94.8% by using the Support Vector Machine technique.

Thiang, Resmana Lim and Andre Teguh Guntoro described car recognition of various car types during daylight and at night based on Gabor wavelets [5]. They extracted Gabor features and created a database of four template images for each vehicle category (sedan, van, pickup). They matched an input image by computing the similarity value to each template. The class of the template with the highest similarity value was chosen. In their experiment, they used the side view of 44 unkown vehicles. Their system achieved an average recognition rate of 93.88%.

2 Preprocessing

Pre-processing the image data means translating the image into a more useful format. Figure 2.1 shows an input image without any pre-processing steps.
The first important step is to determine the position of the vehicles given by the

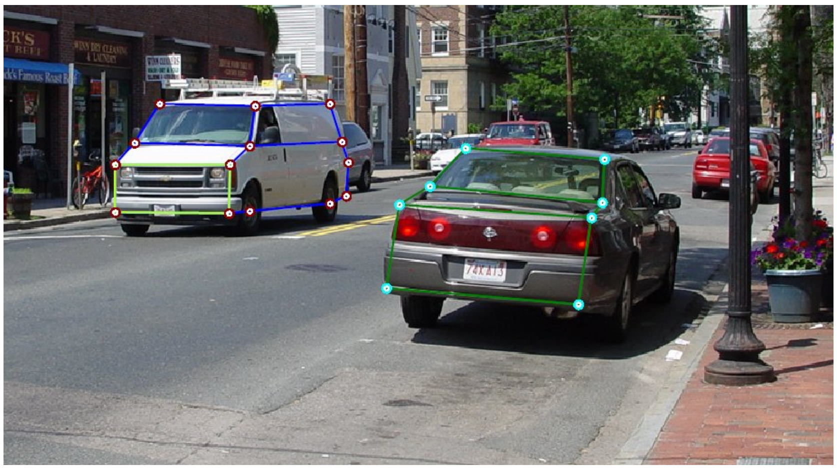


Figure 2.1: Input image with vehicle shapes

landmark points $((x_1, y_1), \ldots, (x_k, y_k))$. As described in chapter 1.3, the determination of the object position is done by another pre-processing system and is not part of this work. We assume that we have an input image as shown in figure 2.1 and a given vector $v_i = ((x_1, y_1), \ldots, (x_k, y_k)$ which describes the position of each vehicle i.
The next step is to extract the detected vehicle i from the input image, so that we can expect one vehicle per image. This is done by finding a rectangle R which contains all point vectors v_i of the corresponding vehicle i.
In the following section, several pre-processing steps are introduced.

2.1 Normalizing

The normalization of an image is an important step to reduce errors caused by image noise, image peaks and differences in contrast and illumination of images.

2.1.1 Binary Images

A simple approach is to use binary images where every pixel of a grayscale image $I(x, y)$ is mapped to a binary value by using a threshold τ:

$$f(I(x,y)) = \begin{cases} 1, & I(x,y) \geq \tau \\ 0, & I(x,y) < \tau \end{cases}$$

This approach is simple, but vulnerable to images noise. Thus, this approach is only useful in combination with other pre-processing methods. In our case, we use this method to reduce the number of edges caused by image noise.

2.1.2 Histogram Equalization

Another more suitable approach is to normalize the image using an image histogram equalization. This approach increases the image contrast by increasing the dynamic range of gray levels. The following briefly introduces the histogram equalization as described in "Digital Image Processing" [6].

Let us consider a greyscale image with discrete pixel values. The probability of the occurrence of gray level k is defined as:

$$p(k) = \frac{n_k}{n} \qquad k = 0, \ldots, L-1 \qquad 0 \leq x_k \leq 1$$

Where L is the total number of gray levels, n the total number of pixels and n_k the number of occurrences of the gray level k.
The so called cumulative distribution function (CDF) of a pixel with gray level

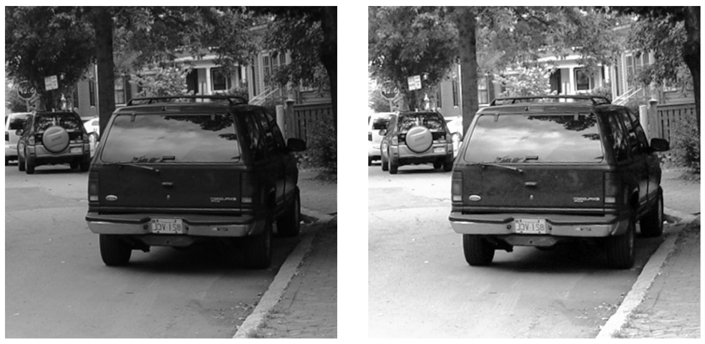

(a) Original greyscale image (b) Histogram equalized image

Figure 2.2: Difference between the histogram equalized image and the original greyscale image

k is defined as the sum of its related probabilities:

$$c(k) = \sum_{j=0}^{k} p(k) \qquad k = 0, \ldots, L-1 \qquad 0 \leq x_k \leq 1$$

In order to equalize a given histogram we apply a transformation which maps each exiting level k to a new level k' so that the CDF of k' will be linearized across its value range:

$$k' = c(k)$$

The values are mapped into an interval $[0, 1]$. To get the original graylevels which are in the interval $[\alpha, \beta]$, the following transformation needs to be applied:

$$k = k'(\beta - \alpha) * \alpha$$

Finally we create a normalized image $I_{Normalized}$ by mapping the pixel values to the normalized gray level distribution using the CDF of $I_{Original}$. The difference between a histogram equalized image and the original greyscale image is shown in figure 2.2.

2.2 Alignment

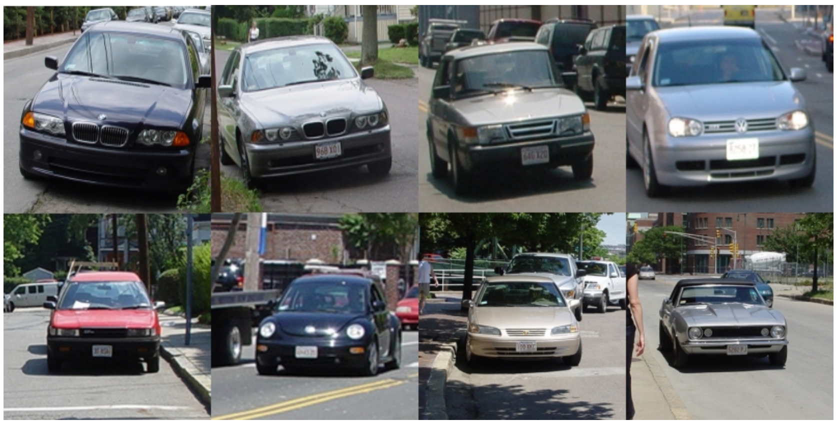


Figure 2.3: Non-aligned images at the top versus aligned images at the bottom

After we have estimated the position of the vehicle we extract every vehicle into a separate image. In order to compare those images or at least to find similar image properties, we have to ensure that the vehicle position, rotation and size is equal on every image. Figure 2.3 illustrates the difference between non-aligned images at the top and aligned images at the bottom. The following sections will introduce two approaches which can be used to align objects in images.

2.2.1 Procrustean Analysis

The first approach is a statistical shape analysis where a given object is described using 2-dimensional coordinates $(x_1, y_1), \ldots, (x_k, y_k)$ which are called landmark points. As described in chapter 1.3, the vehicle detector determines those landmarks points and thus we assume that the parameters $(x_1, y_1), \ldots, (x_k, y_k)$ are given. The next step is to remove translational, rotational and scaling components from the landmark points.
The translational component can be removed by shifting each coordinate so that the mean of all coordinates lie at the origin. The mean of coordinates is given as follows:

$$\bar{x} = (\sum_{i=1}^{k} x_i)^{\frac{1}{k}} \qquad \bar{y} = (\sum_{i=1}^{k} y_i)^{\frac{1}{k}}$$

Finally we map each coordinate onto its normalized value:

$$\phi_{translation}(x, y) = (x - \bar{x}, y - \bar{y})$$

In order to remove the scale component, we have to determine the size s of the object shape:

$$s = \sqrt{\sum_{i=1}^{k} (x_i - \bar{x})^2 + (y_i - \bar{y})^2}$$

Now we can normalize the coordinates by using this scale:

$$\phi_{scale}(x, y) = (\frac{x_1 - \bar{x}}{s}, \frac{y_1 - \bar{y}}{s})$$

To remove the rotational component is a bit more complex. Let us consider two objects with the coordinates $((x_1, y_1), \ldots, (x_k, y_k))$ and $((u_1, v_1), \ldots, (u_k, v_k))$. The rotational difference is at a minimum if the Procrustes distance d is at a minimum, too. The Procrustes distance d of the given objects is defined as:

$$d = \sqrt{\sum_{i=1}^{k} (u_i - x_i)^2 + (v_i - y_i)^2}$$

So we take the coordinates of the first image and look for an angle θ which minimizes the Procrustes distance d, where the coordinates of the second image are changed according to: $(u_1, v_1) = (\cos\theta u_1 - \sin\theta v_1, \sin\theta u_1 + \cos\theta v_1)$. This search can be done for example by using a least squares technique. The final mapping to remove the rotational component is then given as:

$$\phi_{rotation}(x, y) = (\cos\theta x - \sin\theta y, \sin\theta x + \cos\theta y)$$

2.2.2 Similarity Transform

In "Image Alignment and Stiching", R. Szeliski introduces several motion models to describe the relationship between images and he shows how to parameterize the motion models in order to align an image [7].
Related to this work, the similarity transform is a so called parametric motion model where we try to align one image to another by using a rotation matrix $R \in \mathbb{R}^{3x3}$, a translation vector $t \in \mathbb{R}^3$ and a scaling factor $s \in \mathbb{R}$. Those parameters can be used to establish a mathematical relationship that maps pixels from one image to another.
The similarity transformation can easily be seen as a transformation matrix H:

$$H = sR + t = [sR|t]$$

The corresponding image will be aligned by multiplying it with the parameterized transformation matrix. The challenge of this approach is to determine the parameters for H.
As mentioned in chapter 1.3, the pre-processing system already provides keypoints which can be used for a feature based parameter registration. Accordding R. Szeliski, SIFT features perform the best for getting interesting landmark points [7].
After we extracted the landmarks, they have to be matched by finding the corresponding points on the other image. The simplest way is to compare all feature points from one image against the points of the other one. This approach is quadratic in the expected number of features. But the mentioned pre-processing system returns a minimum number of keypoints and thus, the number of compares is low and the system still performs well.
After we got the correspondences, we can estimate the motion parameters R, t and s. The usual way to do this, is to use the least squares technique and look for parameters which minimize the following equation:

$$E = \sum_i ||Hx_i - x_i'||^2 \tag{2.1}$$

Where x_i and x_i' are correspondences between the found feature points of the different images. The similarity transform between two images has linear relationships between motion and parameters [7]. Thus we can use a simple linear regression to find the best parameters.
In chapter 2.1, we assume that all feature points are matched with the same accuracy. This is not always possible, because some points might be located in more textured regions than other. To handle this, we introduce a variance estimation σ_i^2 for correspondence i and minimize the weighted least squares instead:

$$E = \sum_i \sigma_i^2 ||Hx_i - x_i'||^2$$

2.3 Masking

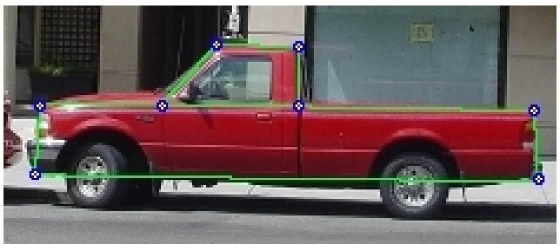


Figure 2.4: Vehicle image extracted from the position which was determined by the pre-processing system

Next to the normalizing and alignment of the image, the masking is also an important part to increase the recognition accuracy. As we see in figure 2.4, the shape returned by the pre-processing system is not accurate and there is still some background left on the extracted vehicle. Thus we need a method to efficiently remove this background noise from the input image. This can be done by applying a grayscale mask $M(x, y)$ to the input image $I(x, y)$ by simply multiplying the pixel values:

$$I(x,y) = \frac{M(x,y)I(x,y)}{\alpha}$$

Where $[0, \alpha]$ defines the pixel value range of the mask. Pixel that fall in dark regions of the mask will get less weight corresponding to the pixel values of the mask.
In figure 2.5 we see different masks which are used in our experiments.
The linear mask shown in figure 2.5(a) gives pixels which are close to the center more weight and let the other pixels disappear:

$$I(x,y) = I(x,y)\frac{-\frac{x_{max}}{2\alpha} * d + \alpha}{\alpha}$$

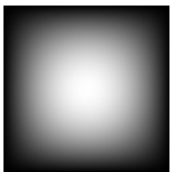
Where d is the distance of the corresponding pixel $p = (x, y)$ to the center $c = (\frac{x_{max}}{2}, \frac{y_{max}}{2})$:

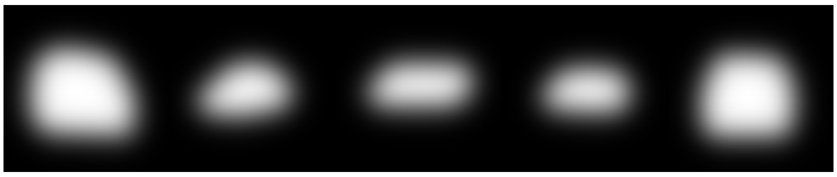
$$d = ||p - c||_2$$

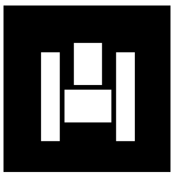
This mask can be applied to any view of the vehicle but a perfect elimination of the background noise is not given and this mask eliminates some parts of the vehicle, too.
In figure 2.5(b), we see a mask for each specific view of the vehicle beginning from the front view, left front, side view, left side and back view. We use the view index given by the pre-processing system and apply the mask $M^k(x, y)$ correspondingly to the current view k:

$$I(x,y) = I(x,y)\frac{M^k(x,y)I(x,y)}{\alpha}$$

To further improve performance, we can reduce the input data by eliminating uninteresting parts of the vehicle itself. We confine the feature extraction to the most characteristic parts on the vehicle defined by rectangles of interest like shown in figure 2.5(c) for a single view and in figure 2.5(d) for multiple views. In the evaluation, we will see that the last method performs best of all masks. The challenge is to find the optimal rectangle positions. The pre-processing system provides us keypoints which define the vehicle position so that we can determine the rectangle positions relative to this. We will discuss some approaches to optimize the mask parameters in chapter 6.3. The rectangle positions and sizes which are used in our experiments can be found in appendix A.3.


(a) Linear, view-independent mask


(b) Non-linear, view-dependent masks


(c) View-independent, partial mask

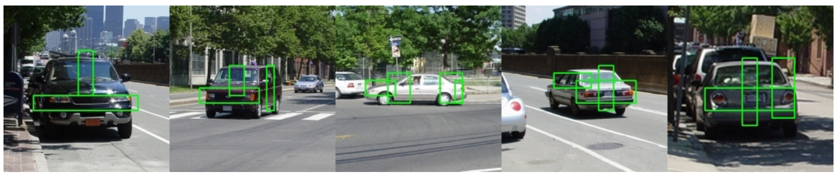

(d) View-dependent, partial masks

Figure 2.5: Different masks applied to input images

3 Methods

3.1 Feature Extraction

Before we are able to classify an object, we have to extract the object parameters. To stay performant, our intention is to keep the size of parameters as small as possible. The transformation of input data into a reduced, representative set of features is called feature extraction. The data structure containing the features is called feature vector or descriptor.
In regard of the problem formulation, we need a feature vector which describes the outline of the vehicle and attributes which are typical for the corresponding car type, respectively. As described in chapter 1.3, the detection system returns the view index and the position of the vehicle. This information is used to apply pre-processing steps like described in the next chapter 2. Due to those processing steps, we can assume that our input image $I(x, y)$ has only a negligible amount of background noise. The following sections introduce several approaches for feature extraction.

3.1.1 Laplace of Gaussian

The first approach tries to transform the outline of the vehicle into a feature vector by using the Laplace of Gaussian operator:

$$LoG(x, y) = \nabla^2(G(x, y) * I(x, y))$$

Where $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ is the Laplace filter for a 2D function f. We implement this second-order derivate as a filer matrix with a kernel aperture size of 1. As a approximation of the filter, we might use the following filter mask:

$$\nabla^2 \approx \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

The Laplace filter is known to be very sensitive to noise [6]. So we're using the Gaussian filter $G(x, y)$ as a low-pass filter in order to smooth the image and remove noise before we start to detect edges. The feature vector of an 128×128 dimensional image $I(x, y)$ is formed by $LoG(x, y)$ and has the dimension $n = 128 * 128$.

3.1.2 Gabor Wavelets Transform

The Gabor wavelets transform $O_k(x, y)$ of a grayscale image $I(x, y)$ is the convolution with a set of wavelets $\psi_k(x, y)$:

$$O_k(x, y) = I(x, y) * \psi_k(x, y)$$

The wavelets are formed by a set of plane waves with different translations, rotations and sizes restricted by a Gaussian envelope function:

$$\psi_k(z) = \frac{\|k\|^2}{\sigma^2} e^{\frac{-\|k\|^2 \|z\|^2}{2\sigma^2}} [e^{ikz} - e^{\frac{-\sigma^2}{2}}]$$

With $\sigma = 2\pi$, $k(\mu, \upsilon) = \frac{k_{max}}{f^{\upsilon}} e^{i\pi\frac{\psi}{8}}$ (with the maximum frequency k_{max}), spacing factor $f = \sqrt{2}$, scale υ and orientation μ. Good test result using Gabor wavelets can be obtained by a wavelet family with five different scales $\upsilon \in \{0, \dots, 4\}$ and eight orientations $\mu \in \{0, \dots, 7\}$ as shown in figure 3.1 [3] [8].
The resulting feature vector g is formed by $O_k(x, y)$. So if we use 128×128 dimensional images and a down-sample factor of 64, we get a final feature vector with a length of $n = \frac{128*128*5*8}{64} = 10240$ elements.
In order to increase the performance, the Gabor wavelets can be computed in a pre-processing step. Since the image size does not differ, we can use the same wavelet database to extract feature from several images.

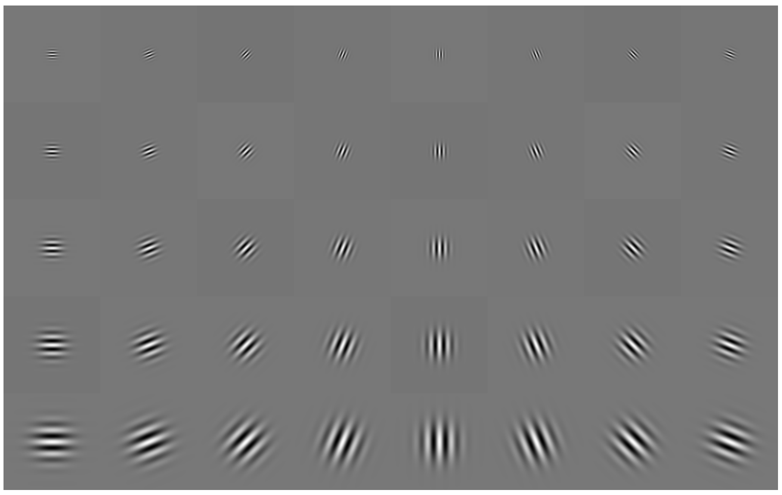


Figure 3.1: Gabor wavelets with different scales and orientations

3.1.3 Histogram of Oriented Gradients

The basic idea of the histogram of orientated gradients (HoG) is to characterize appearance and shape of an object by using the distribution of local edge direc-

tions. Navneet Dalal and Bill Triggs first used the HoG extraction method in their paper "Histograms of Oriented Gradients for Human Detection" [9].
To build an so called HoG image, we split the image into connected regions called cells and compute the gradient directions or edge orientations for each pixel. For each cell, we create an orientation based histogram where the pixel gradients of the current cell are accumulated corresponding to their orientation. Hence, each cell $c_i \in \mathbb{R}^n$ is a vector which contains the magnitudes for n different orientations. To boost this accumulation, we need an efficient method to calculate rectangular sums of the gradient values. By using integral images we can compute any rectangular sum in four array references [10]. The integral image $II(x, y)$ of an input image $I(x, y)$ at position x, y contains the sum of pixels above and left to x, y:

$$II(x, y) = \sum_{x_1 \leq x, y_1 \leq y} I(x_1, y_1)$$

In our case, we can compute an integral image containing the corresponding gradient for each pixel. We can use this image to efficiently sum up those values for each cell like shown in figure 3.2.
To further improve the performance and to achieve greater invariance in illumi-

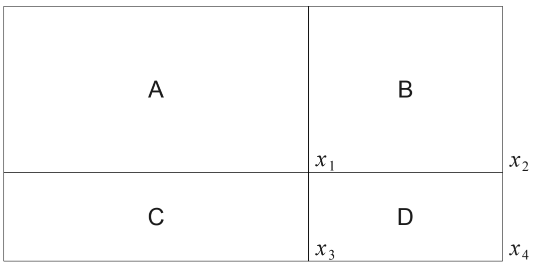


Figure 3.2: Using integral images, the rectangle sum of D can be computed in four array references: $D = x_4 - x_3 - x_2 + x_1$

nation and shadowing, we use block normalization where we further group each cell into larger regions called blocks. The blocks are placed in a way, so that each cell is shared by four overlapping blocks. For each block B, we normalize the cells $c_i \in B$ by using the l_2- norm:

$$c_i = \frac{c_i}{\sqrt{\sum_{Cell\, c} c^2}}$$

In our implementation, the normalized gradient values are subjected to a threshold. In this case, another normalization is necessary after thresholding.
For an image with the size of $w \times h$ pixels, the normalized image gradients for each orientation form the final descriptor with size $\left(\frac{w}{c_w} - 1\right)\left(\frac{h}{c_h} - 1\right) * 4 * n$, where c_w and c_h stand for the cell size and n for the number of different orientations.
Figure 3.1.3 shows an HoG descriptor where the orientation for each cell is visualized and the gradient magnitude is defined by the corresponding grey level.

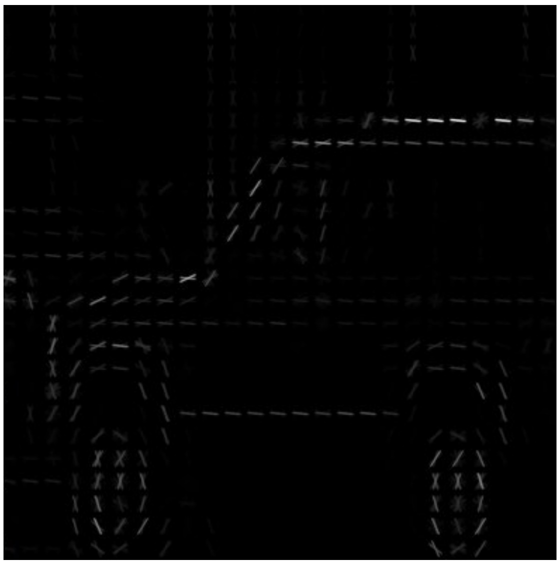


Figure 3.3: HoG image of a Jeep

3.1.4 Scale Invariant Feature Transform

The Scale Invariant Feature Transform (SIFT) transforms an image into vectors which are invariant against translation, rotation and scaling. This approach was first introduced by David G. Lowe in "Object Recognition from Local Scale-Invariant Features" [11] and "Distinctive Image Features from Scale-Invariant Keypoints" [12].

The first step of SIFT is to identify key locations and scales that can be repeatably assigned under differing views of the same object. That means, we are looking for stable features across all possible scales σ in image $I(x, y)$ by using a continuous function of scale. Such a function is called scale space:

$$L(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} * I(x, y)$$

The key locations we are looking for are the extremes of a different-of-Gaussian function $D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$.
An efficient approach to compute $D(x, y, \sigma)$ is to create a set of Gaussian images with 3 different scales by using $L(x, y, k\sigma)$ with $k = \sqrt{2}$ and $\sigma = 1.6$. This set of Gaussian images is called octave G_0. Further, we compute the difference-of-Gaussian between the images of the first octave G_0 and downsample the resulting images by a factor of 2. Those down-sampled Gaussian images form the next octave G_1. We repeat the whole process with a doubled scale $\sigma = 2\sigma$ and the constant $k = \sqrt{2}$ until we got at least 3 resulting Gaussian images.
To find the maximum and minimum of a pixel, we compare its eight neighbours at the current octave and the nine neighbours at the octave above and below. The pixel is selected only if its value is larger (smaller) than all neighbour values.

This approach gives us a considerable number of keypoint candidates. A lot of those locations are sensitive to noise or poorly located along an edge. To remove those unstable keypoint candidates, we apply a threshold on minimum contrast and on ratio of principal curvatures to eliminate edge responses. The remaining locations form the set of keypoints.
To assign an orientation to a keypoint, we compute the gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ for each sample $L(x, y, \sigma) = L(x, y)$ where σ is the scale of the keypoint:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+q) - L(x, y-1))^2}$$

$$\theta(x, y) = \arctan \frac{(L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}$$

We compute those values for a set of pixels within a region around the keypoint and create an orientation histogram which covers 360 degree by using the orientation $\theta(x, y)$ and weight $m(x, y)$ of each sample point. The orientation of the highest peak and of other local peaks close to that one is assigned to the point. Thus multiple orientations with different magnitudes are possible. In order to achieve invariance in orientation, the gradient orientations are given relative to the keypoint orientation. Finally, the gradients of the sample points are weighted by a circular Gaussian window. The center of this window is the keypoint location itself. Gradients that are far away from the center achieve less emphasis.
In the final step, we divide the region around the keypoint location into further sub-windows and create another orientation histogram for each new sub-window. This time, the histogram has only eight bins and accumulates the gradient weights for the corresponding orientation. The accumulated gradient weights of each orientation and sub-window form the final descriptor.
Figure 3.4 shows a visualization of keypoints from a side view of the vehicle. The arrows indicate the keypoint orientation and magnitude with the highest peak.

3.2 Classifier Design

In order to classify input data, we have to group similar feature vectors together. The classifier assigns one of the existing classes to an unseen feature vector based on an underlying classification model. We can differ between several classification models.
The generative model first determines the conditional density functions $p(v|C_k)$ and the prior class probability $p(C_k)$ for each class k and feature vector v. By using the Bayesian rule, we can now determine the post probability $p(C_k|v) = \frac{p(v|C_k)*p(C_k)}{p(v)}$. The vector v is assigned to the class C_k for which $p(C_k|v)$ is the maximum. This model is used for example in the k-nearest neighbour algorithm

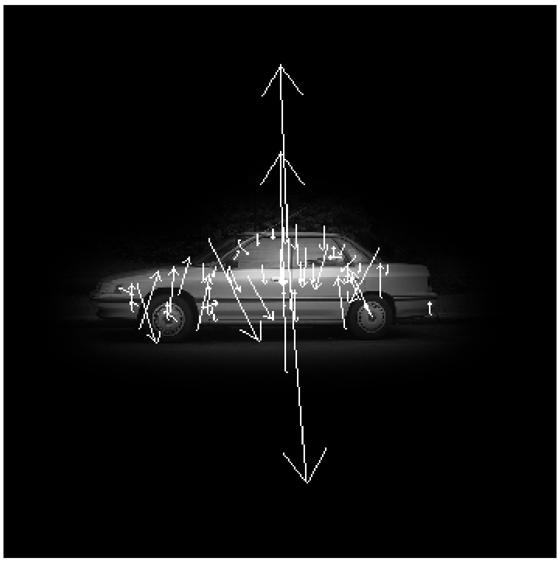


Figure 3.4: Masked side view with visualized keypoints

which is briefly introduced in chapter 3.2.1.
In the discriminative model, the dependence of the input feature vector v is modeled on observed training data. The conditional probability destribution of the input vector v is directly computed from the training set. Other than the generative models, the discriminative models do not allow the use of the joint probability distribution. We will introduce a discriminative model called Support Vector Machines in chapter 3.2.2.

3.2.1 k-nearest Neighbour

The k-nearest neighbour algorithm (k-NN) is a quiet simple approach in machine learning. Given an input vector, this approach gets the classes of the k-nearest neighbors. The most common class within those neighbors is assigned to the input vector.
To get the nearest neighbor of an image, we just calculate the difference of all pixel values from the input image $I(x, y)$ and the labeled image $T_k(x, y)$ of class k:

$$D_k = \sum_{x,y}(I(x, y) - T_k(x, y))$$

To classify an image, we assign the most common class of the k-nearest neighbors, which are the template images for which D_k is a minimum. We'll see in chapter 4 that the practical applicability is limited and the computation of the k-nearest neighbours is time consuming and leds to poor performance.

3.2.2 Support Vector Machines

A more efficient approach is to use a Support Vector Machine trained on a given training set of label pairs (x_i, y_i), where $x_i \in R^n$ and $y_i \in \{-1, +1\}$ for $i = 1, \ldots, l$. The following section gives an introduction to Support Vector Machines. The following definitions are based on "A Tutorial on Support Vector Machines for Pattern Recognition" by Christopher J.C. Burges [13].

Separable Case

Let us assert, that the given feature space R^n is separable. Thus we can find an hyperplane which separates the training samples based on their labels. To find the best solution, we want to look for the hyperplane which maximizes the margin $d_+ + d_-$ between the closest points of each class. In figure 3.5 we see a linear separable case, where all training samples follow the constrains:

$$\begin{aligned} x_i w + b &\geq +1 \quad for\, y_i = +1 & (3.1) \\ x_i w + b &\leq -1 \quad for\, y_i = -1 & (3.2) \end{aligned}$$

If we combine 3.1 and 3.2 we got the following:

$$y_i(x_i w + b) - 1 \geq 0 \quad \forall i \tag{3.3}$$

All training samples for which this equation is 0 are called Support Vectors and lie on one of the hyperplanes H_1, H_2

$$\begin{aligned} y_i(x_i w + b) - 1 &= 0 \\ \Rightarrow H_1 : x_i w + b &= +1 \\ \Rightarrow H_2 : x_i w + b &= -1 \end{aligned}$$

The margin $d_+ + d_-$, which has to be maximized changes to $d_+ + d_- = \frac{1}{\|w\|} + \frac{1}{\|w\|} = \frac{2}{\|w\|}$. Where w is normal to the hyperplane and $\|w\|$ is the Euclid distance of the norm.
We see that the hyperplanes H_1 and H_2 are in parallel and there are no training samples between. Thus there exists an hyperplane H for which the margin is maximal. The maximal margin can be achieved by minimizing $\|w\|$. In order to handle this optimization problem, we convert it into the Lagrangian formulation, which is easier to handle and which allows the generalization to the non-linear case in chapter 3.2.2.

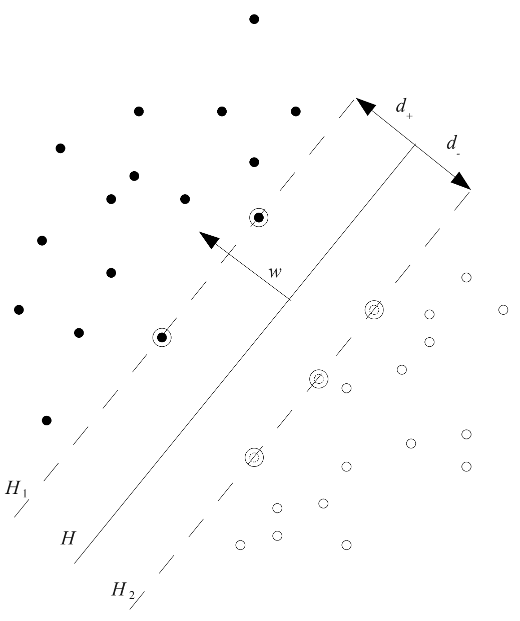


Figure 3.5: Example of a linear separable feature space

Lagrange Formulation

After we introduced the Lagrange multipliers α_i for $i = 1, \ldots, l$, we can convert the problem formulation to the following:

$$L_p = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i w + b) + \sum_{i=1}^{l} \alpha_i \tag{3.4}$$

Where L_p is to minimize in respect to w and b while the derivates of L_p vanish and $\alpha_i \geq 0$. This is a convex, quadratic problem so that we can equivalently solve the dual problem where L_p has to be maximized in respect to w and b and $\alpha_i \geq 0$. Again, derivates of $L_p$vanish. Thus we can form the following equations:

$$w = \sum_i \alpha_i y_i * x_i \tag{3.5}$$

$$\sum_i \alpha_i y_i = 0 \tag{3.6}$$

Using 3.5 and 3.6, we can form the dual problem as the following:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i * x_j \tag{3.7}$$

As we see, there's a Lagrange multiplier α_i for every training sample (x_i, y_i). Support vectors are the points which lie on one of the hyperplanes H_1 and H_2 and for which $\alpha_i > 0$. Those support vectors are the critical elements of a training set. If the feature space is difficult to separate, a lot samples are located along the hyperplanes H_1 and H_2 which results in an high number of support vectors. Once we've trained a SVM, we can us it by simply determining on which side of the hyperplane a given sample lies.

Non-Separable Case

In the non-separable case, we would like to relax the conditions 3.1 and 3.2 by introducing slack variables ξ_i for $i = 1, \ldots, l$:

$$x_i w + b \geq +1 - \xi_i \quad for\, y_i = +1 \tag{3.8}$$

$$x_i w + b \leq -1 - \xi_i \quad for\, y_i = -1 \tag{3.9}$$

$$\xi_i \geq 0 \quad \forall i$$

$\sum_i \xi_i$ becomes an upper bound of the number of training errors. By replacing 3.8 and 3.9 in the dual problem formulation 3.7, we got the following:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i * x_j \tag{3.10}$$

With $0 \leq \alpha_i \leq C$, $\sum_i \alpha_i y_i = 0$ and $w = \sum_{i=1}^{Ns} \alpha_i y_i x_i$ where Ns is the number of support vectors. This equation is the same like 3.7 except that C is now an upper bound for α_i. Thus a larger C means an higher error penalty for the separation problem. So the user can define the error penalty by defining parameter C.
The primal Lagrange formulation for the non-separable case is now:

$$L_p = \frac{1}{2}\|w\|^2 + C \sum_i \xi_i - \sum_i \alpha_i (y_i (x_i w + b) - 1 + \xi_i) - \sum_i \mu_i \xi_i \tag{3.11}$$

Where μ_i are the Lagrange multipliers which enforce the positivity of ξ_i.

Non-Linear Case

B. Schohlkopf and A. Smola have shown, that the number of possibilities to separate a non-linear problem can be increased by mapping the feature space onto an high dimensional space $\mathcal{H}$ [14]. Thus we introduce a mapping $\phi : R^n \rightarrow \mathcal{H}$ and kernel functions $K(x_i, x_j) = \phi(x_i)\phi(x_j)$. In order to support non-linear separable problem, we replace in the training algorithm the dot products $x_i * x_j$ with the kernel functions $\phi(x_i)\phi(x_j)$. Now, the corresponding Lagrange formulation looks like:

$$\begin{aligned} L_D &= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \phi(x_i)\phi(x_j) && (3.12) \\ &= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j K(x_i, x_j) && (3.13) \end{aligned}$$

If we choose suitable kernel functions $K(x_i, x_j)$ we don't need to work with ϕ explicitly. Also for testing a sample x, we can avoid the explicit computing of ϕ:

$$\sum_{i=1}^{Ns} \alpha_i y_i \phi(s_i)\phi(x) + b = \sum_{i=1}^{Ns} \alpha_i y_i K(s_i, x) + b$$

where s_i are the support vectors.
As we see, we can still do a linear separation but in a different space. By using suitable kernel functions, we can map a non-linear separable problem onto an higher dimensional space where it might be easier to separate. To validate kernels for which there is an high dimensional space $\mathcal{H}$ and a mapping ϕ so that the training will converge, the Mercer's Condition is sufficient [15].
The following SVM kernels are supported by the LIBSVM software which is used in our experiments in chapter 4.

$$\begin{aligned} (linear) \quad K(x, y) &= x * y && (3.14) \\ (polynomial) \quad K(x, y) &= (\gamma x * y + r)^d && (3.15) \\ (radial) \quad K(x, y) &= e^{-\gamma\|x-y\|^2}, \gamma > 0 && (3.16) \\ (sigmoid) \quad K(x, y) &= \tanh(\gamma x * y + r) && (3.17) \end{aligned}$$

The parameters γ, r and d are kernel parameters and have to be defined by the user. These parameters can be optimized in order to achieve a greater accuracy.

4 Experiments

4.1 Training and Testing Sets

In order to compare the different methods, we are using an image database of 5461 aligned images where the view index and the vehicle type is given as label. Table 4.1 shows the distribution of the images.
All images were taken in a public domain without regard to any obstacles like trees, people or other vehicles covering the direct view to the vehicle.
To classify the data, we use both, the k-nearest neighbour algorithm described

View	Number of Images
Head	916
Left Head	1393
Left Side	798
Left Tail	1219
Tail	1135

Table 4.1: Distribution of images

in chapter 3.2.1, as well a SVM as described in chapter 3.2.2.

In case of k-nearest neighbour, we evaluate the methods by using a v-fold cross validation with $v = 5$. Thus we split the data into five subsets and test each subset against the other $v - 1$ subsets. For each test sample, we simply look for the k-nearest neighbour and get the most common label within those neighbours. The accuracy is the average percentage of correct classified samples of all testing sets.

When using a SVM for recognition, we train classification models and use cross validation, too. The following notes are based on "A Practical Guide to Support Vector Classification" by Chih-Wei Hsu, Chih-Chung Chang and Chih-Jen Lin and describe the SVM setup for our experiments [16].
Before we train the data by a SVM, it is recommended to scale the data in order to avoid feature values in greater numeric range dominate those in smaller ones. In our approaches, we scale both training and testing data to the range $[-1, 1]$.
Further we must handle feature vectors in a non-linear separable feature space, so we need to follow the non-linear separable case described in chapter 3.2.2. It

is recommended to use the radial basis function 3.16 as kernel function. This kernel fulfills the requirements to map the non-linear feature space onto an higher dimensional space while having less numerical difficulties and less hyperparameters as for example the polynomial kernel 3.15.
To determine the kernel parameter γ and the error penality C, we use a simple grid search algorithm. This algorithm trains and tests the data while using different values for γ and C. Finally the parameter which achieve the best accuracy is chosen. For further improvement of the accuracy, we run this grid search twice. In the first run, the parameter $\log_2(\gamma)$ changes its value between $[-5, 15]$ with step size of 2 and $\log_2(C)$ changes its value between $[3, -15]$ with a step size of -2. After this run is finished, we get the optimal parameters for γ and C within this grid and start another grid search with same error penality C but with a smaller step size and a smaller interval around γ. A table with all SVM parameters which were used in our experiments can be found in appendix A.2.
To prevent the overfitting problem, we use $v-$fold cross-validation where we split the training data into v subsets of equal sizes. A classification model is trained by using the $v - 1$ subsets. Finally, the classification model is tested with the remaining subset. The accuracy is the average percentage of the correct classified data. In the following we use a folding value of $v = 5$.
In order to evaluate the test results, we use the recognition accuracy we got from v-fold cross validation and the number of support vectors of the classification model. A table with classification rates and number of support vectors can be found in appendix A.1.

4.2 Accuracy

4.2.1 Introduction

We can split the evaluation of the experiments into several parts.
The first part is the feature extraction which returns the feature vector v. This vector can be seen as a numbered list of values:

$$v = [(1, v_i), (2, v_2), \ldots]$$

This part is one of the most time-consuming parts. Thus we provide a C/C++ application for each kind of feature, which uses the Open Computer Vision Library (OpenCV) [17].
The next part scales the feature values and optimizes the kernel parameters by using a grid search. Finally the average accuracy is determined by using v-fold cross validation. Those parts are done by libsvm which is provided by Chih-Wei Hsu, Chih-Chung Chang and Chih-Jen Lin [16].

4.2.2 Laplacian of Gaussian

In order to give the same weight to all parts of the vehicle, we first equalize the greyscale image by using histogram equalization.
Textured images lead to an high edge detection rate. Therefore, this approach is very sensible to background noise and we have to extract the vehicle as accurate as possible. Due to this sensibility, we use the partial view-dependent mask in figure 2.5(d).
Finally, we filter the masked image parts by Laplace with an aperture kernel size of 1. The pixel values of the resulting LoG image form the feature vector.
After we created a feature database with all images, we can start the classification. We divide the set of feature vectors into v subsets and run a v-fold cross validation by using k-nearest neighbour as classifier.

Table 4.2.2 shows the recognition accuracy of this approach for each view and

k	View 0	View 1	View 2	View 3	View 4
1	68,45	70,2	68,8	16,57	76,65
2	**72,71**	**72,93**	68,8	48,56	**76,39**
3	**72,71**	71,86	**69,55**	37,24	76,38
4	72,6	71,86	64,16	**51,35**	**76,39**
5	72,6	71,79	68,55	50,04	**76,39**
6	72,6	71,93	65,91	46,1	**76,39**
7	72,6	71,86	69,17	39,62	**76,39**
8	72,6	71,86	68,8	38,56	76,38
9	72,6	71,86	69,17	49,14	**76,39**
10	72,6	71,93	69,54	45,78	**76,39**

Table 4.2: k-nearest neighbour classification accuracy of LoG features in %

different k-nearest neighbours. We see that the tail view is the best classified view and the recognition of the left tail view is the worst. We will see in the other experiments that the side view is best classified. This effect might arise from irregular textures caused by image interference and reflections. In spite of the Gaussian blur, the Laplace filter detects more edges than only those of the vehicle shape.

An visualized feature vector is given in image 4.1. We can see, that reflections lead to a very imprecise vehicle shape. Thus we need to find an extraction method which is more resistant against image interference like reflections or coarse textures and which returns a more accurate vehicle shape description.

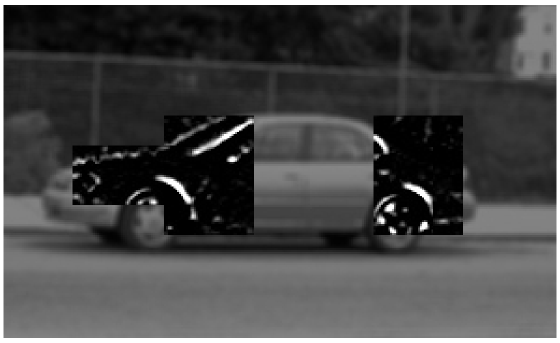


Figure 4.1: Partial laplace filtered image

4.2.3 Image Texture

A quiet different approach is to use only texture information and classify the vehicle based on its characteristic textures.
The naive method is to mask the histogram equalized image dependent on the view and extract the pixel values which form the feature vector. We resize the image to 64×64 pixels which results in a feature vector size of 4096. We can see in table 4.2.3 that we still get a poor accuracy. The reason might be background noise which still exists around the vehicle. Another reason might be that parts of the vehicles have different colors. Although we use grayscale and histogram equalized images, the pixel values of different colored parts still differ.
In the next experiment, we use the partial view-dependent mask and try to classify the pixel values of the ROIs. The results are shown in table 4.2.3. Again, we achieve a weak classification accuracy. Finding characteristic, color independent parts turns out to be difficult due to a large number of varying vehicle types. This results in a feature space that is difficult to separate.

View	Support Vectors	Accuracy in %
0	444	**87.01**
1	851	79.33
2	490	73.93
3	669	78.01
4	576	83.88

Table 4.3: SVM classification result of texture using a view-dependent mask

View	Support Vectors	Accuracy in %
0	596	72.60
1	931	71.93
2	589	68.42
3	830	72.60
4	636	**76.30**

Table 4.4: SVM classification result of texture from view-dependent ROIs

4.2.4 Histogram of Orientated Gradients

Following the last evaluations, we need a feature extraction method which is robust against local noise. Thus, we combine several pixels to a cell and we compute the average orientation degree of that cell.
Again, we first run an histogram equalization of the image in order to improve the contrast. Further, we want to create features which describe the whole vehicle shape. Due to the block normalization of the HoG method, we do not have to worry about local noise and we can use the view-dependent mask from figure 2.5(b).
By using HoG, the 256×256 grayscale, histogram normalized image $I(x, y)$ is split into blocks with a size of 16×16 pixels. Every block contains $n_{cell} = 4$ cells with a size of 8×8 pixels. Further the block normalization is applied where each cell is shared by 4 overlapping blocks. For each cell, the gradient values for $n_{bin} = 9$ different orientations are computed. Those values form the final feature vector, which has a total size of $(\frac{\max(i)}{c_x} - 1) * (\frac{\max(j)}{c_y} - 1) * n_{cell} * n_{bin} = (\frac{256}{8} - 1) * (\frac{256}{8} - 1) * 4 * 9 = 34596$ features.
In this experiment, we use the SVM which transforms the feature space into an high dimensional space to increase the separation possibilities.
Table 4.2.4 shows the result of a 5-fold cross validation returned by the SVM

View	Support Vectors	Accuracy in %
0	569	78.71
1	938	76.88
2	483	82.58
3	701	82.03
4	580	**85.20**

Table 4.5: SVM classification result of HoG features using a view-dependent mask

with a radial basis kernel and parameters optimized by grid search. As we see, this approach performs better than the previous one. Again, the tail view can be classified the best with 85.02% and 572 support vectors. But if we compare the number of support vectors $n_4 = 572$ against the total number of samples

$s_4 = 1135$, we see that more than the half of all samples are a support vectors. Thus, the HoG features form a feature space which is difficult to separate.
We repeat the experiment, but this time, we use a partial view-dependent mask

View	Support Vectors	Accuracy in %
0	604	79.69
1	624	85.79
2	291	**94.11**
3	699	87.12
4	468	88.81

Table 4.6: SVM classification result of HoG features from view-dependent ROIs

which returns only ROI's so that there's no further background texture. Table 4.2.4 shows the result of this experiment. As we see, the partial mask further increases the accuracy while the number of support vectors is less than with the view-dependent mask. This shows, that the partial mask makes the problem easier to separate. Next to that, we reduce the size of the feature vector by using ROI. For example if we use the partial side mask, we extract patches with sizes 64×48, 64×48, 32×64 and 48×16 thus we get a feature vector size of $\big(2(\frac{64}{8}-1)(\frac{48}{8}-1)+(\frac{32}{8}-1)(\frac{64}{8}-1)+(\frac{48}{8}-1)(\frac{16}{8}-1)\big)*4*9 = 3456$ instead of 34596 like the previous one. This makes the classification process more performant due to a smaller SVM model.
In this experiment, we can see that the side view gets classified best compared to other views. But in the other experiments, the tail view was the best one. That can be caused by the partial feature extraction from the ROIs. Thus, the quality of the extracted features strongly depends on the used ROIs. In order to keep the comparability of the experiments, we will use the same ROI parameters in all experiments. We will discuss some approaches to optimize those parameters in chapter 6.3.

View	Support Vectors	Accuracy in %
0	546	83.30
1	716	87.58
2	293	**95.49**
3	557	89.83
4	520	91.19

Table 4.7: SVM classification result of HoG features with appended texture from view-dependent ROIs

If we think about the head or tail view of a car, we notice, that some parts of the vehicle have a characteristic texture like e.g. the radiator grill. This is why we start another experiment where we append the texture description to the feature vector. This is done by getting the pixel values from ROIs like described in chapter 4.2.3. The size of the feature vector increases to $3456+68*48*2+32*64+48*16 = 12800$. This vector is slightly bigger then the last one, but still smaller than the first one. We still have a relative small SVM model and therefore a performant classifier. Table 4.2.4 shows the result of this experiment. We can see, that the accuracy of every view is better than the accuracy without the texture information while the number of support vectors is only slightly more.

4.2.5 Gabor Wavelets Transform

Next to HoG, we describe in chapter 3.1.2 another method to extract orientation gradients. This method uses Gabor wavelets with different orientation and different scales in order to get a vehicle shape which is more invariant against scale than HoG.
Again, we start with a view dependent mask and a grayscale, histogram equalized

View	Support Vectors	Accuracy in %
0	499	89.41
1	778	85.14
2	403	90.98
3	695	84.00
4	525	**91.80**

Table 4.8: SVM classification result of Gabor features using a view-dependent mask

image with size 128×128. It is generally recommended to use Gabor wavelets with five different scales and eight orientations [3] [8]. Together with a down sample factor of 64, this results in a feature vector length of $\frac{128*128*5*8}{64} = 10240$. The accuracy of this experiment is shown in table 4.2.5. We can see, that the the classification rate for every view is greater than by using HoG features with a view-dependent mask. This implies, that the Gabor feature could be more suitable to describe the vehicle shape.
In order to reduce the feature length, we start another experiment where we extract the Gabor features only for ROIs. We use the same partial view-dependent mask as we used in the HoG experiment. Table 4.2.5 shows the result of this experiment. If we compare the results against the HoG experiment, we see that we get a slightly better accuracy in all views. The improvement might arise from the scale and rotation invariance of the Gabor features.
Like in the other experiments, we append the image texture of the ROIs to the corresponding feature vector and restart the experiment. The result is shown in table 4.2.5. We see, that the number of support vectors slightly increased, but the

View	Support Vectors	Accuracy in %
0	523	85.69
1	809	87.01
2	350	**96.49**
3	613	90.24
4	505	91.10

Table 4.9: SVM classification result of Gabor features from view-dependent ROIs

recognition accuarcy for the head and tail view has been improved, too. Again, the appended texture information provides suitable characteristic details for the vehicle head and tail, but it decreases the recognition accuracy of the side views. As soon as there are parts which contain reflections or shadows, the texture information leads to less accuracy.

View	Support Vectors	Accuracy in %
0	506	90.72
1	809	86.29
2	417	**93.86**
3	656	88.43
4	545	91,72

Table 4.10: SVM classification result of Gabor features with appended texture from view-dependent ROIs

4.2.6 Scale Invariant Features

Previous experiment in this document shows, that we can achieve a better recognition accuracy if we use a shape description which is more invariant against scale. Thus we try SIFT features which are also known to be invariant to scale and rotation [12]. The region around the keypoint location described in chapter 3.1.4 has the size of 16×16 pixels and is divided into 4×4 sub-windows with size of 4×4 pixels. As described, we achieve the magnitude for eight orientation of each sub-window. This results in a feature vector length of $4 * 4 * 8 = 128$.
The position of the keypoints is not always the same and due to image properties, the number of found keypoints can differ, too. In order to make the images comparable, we align the keypoints to a grid whose cells have the size of 16×16 pixels. The final feature vector is created by concatenating the keypoint descriptors beginning from the left upper grid cell. If a grid cell has no assigned descriptor, a dummy descriptor wich zero values is used instead.

Table 4.11 shows the result of this experiment. If we compare the recognition

View	Support Vectors	Accuracy in %
0	622	81.33
1	837	77.17
2	498	83.21
3	775	80.48
4	631	**85.46**

Table 4.11: SVM classification result of SIFT features using a view-dependent mask

accuracy, we see that the SIFT features extracted from an image with applied view dependent mask achieve slightly weaker results than the HoG features. This effect becomes clear if we compare the features of both methods. While the HoG feature vector represents the orientation gradient for each cell, the SIFT feature vector also contains orientation gradients which are aligned to a 16×16 pixels grid. Corresponding to the found keypoints, there might be grid cells without an keypoint descriptor but with a dummy descriptor containing zero values. In contrast, the HoG feature descriptor contains orientations gradients for each cell of the image. From this one can infer, that the SIFT feature vector still contains enough information to achieve a good accuracy, but ignores unnecessary features which might come from image interference.
We try to increase the recognition accuracy by limiting the keypoint search to the ROIs. Thus we apply a partial view-dependent mask and restart the feature extraction. The result of this experiment is shown in table 4.12.
We see, that the recognition accuracy is signitifcantly worse than in the previous

View	Support Vectors	Accuracy in %
0	569	75.11
1	778	72.29
2	356	79.07
3	670	74.49
4	713	**80.18**

Table 4.12: SVM classification result of SIFT features from view-dependent ROIs

experiment. By limiting the keypoint search to the ROIs, the number of found keypoint is very low and thus the feature vector becomes weak and represents less vehicle characteristics. According to that, the SIFT approach needs the whole image of the vehicle in order to achieve good results. But in most cases, this data contains information like reflections, blanketing objects or shadows which falsifies the vehicle description.

4.3 Performance

As we mentioned in chapter 1.2, we only use a single image as input data. This complicates the elimination of the background around the vehicle and makes the recognition problem more difficult. Further, in regard to support sequential images, the vehicle type classification should be fast and should be finished within milliseconds.
The k-nearest neighbour classifier takes the feature vector of the input image and computes the k nearest neighbours to the training samples. Since our recognition problem is quiet difficult, we need a great number of training samples to be able to separate the classes. Further, the k-nearest neighbour classifier performs in $O(n)$ where n is the number of training samples. In order to perform a classification by using a LoG feature space, the computation of the k-nearest neighbours needs about 9210 ms. Thus, this classifier is not qualified to support seqential images if we have to handle a great number n of training samples.
Using a SVM model is more efficient for our needs. The SVM takes the training set and maps the feature space into an higher dimensional space where the problem might become easier to separate. Further, the SVM trains a classification model where only the support vectors are stored. Those vectors are used to describe the hyperplanes which were optimized by a training. To classify an input image, the SVM only needs to compute on which site of the hyperplanes the input vector is located. Finally, the class of the corresponding area is assigned. Thus, this classifier performs in $O(m)$ where m is the number of support vectors. The previous experiments show, that the number of support vectors is vastly smaller than the number of total training samples. Therefor, the SVM is more suitable in order to support a fast classification.
Table 4.3 shows the timing statistics for feature extraction, scaling and recognition by using the corresponding classifier. The performance evaluation was executed on a Intel(R) Core(TM)2 with 2.00 GHz and 2 GBytes DDR2 RAM. We see, that the duration of the recognition strongly depends on the size of the classification model. But before we can perform the recognition, we have to extract and scale the features from the input image. If we look at the Gabor and HoG features extracted from a whole view-masked image, we see that an high dimensional feature vector leads to a great overhead in this process. Thus, in order to support a fast recognition, small feature vectors are recommended. In regard to the recognition accuracy, the Gabor and HoG features extracted from ROI perform the best.

4.4 Conclusion

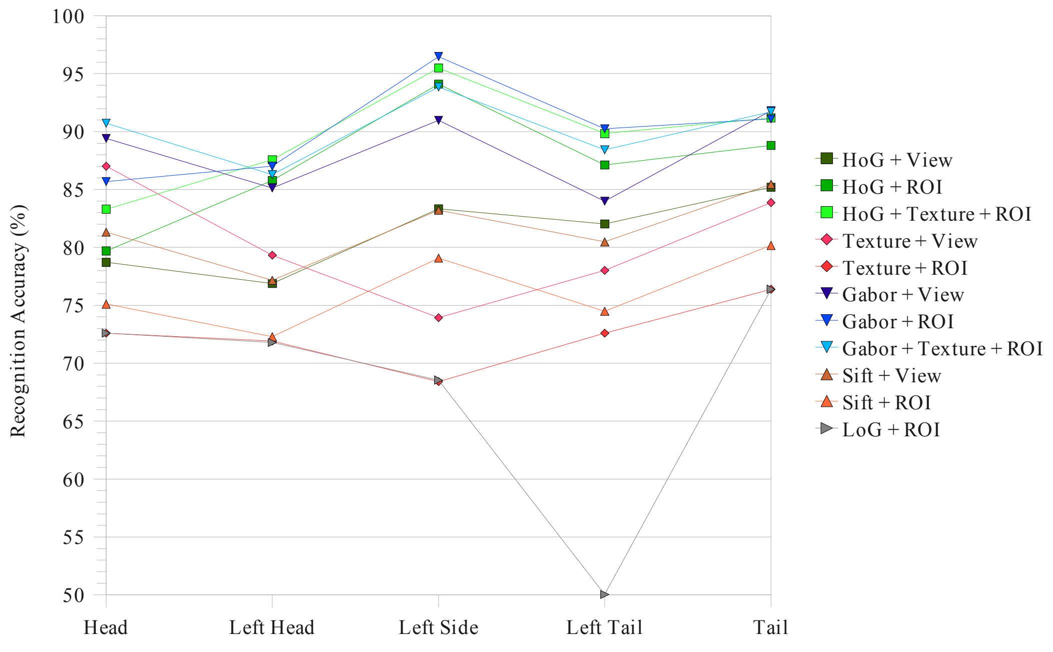
The recognition accuracy for all introduced methods is shown in figure 4.2(a). The number of support vectors for the SVM classifier is shown in figure 4.2(b).
In every experiment except for the Laplacian of Gaussian features (LoG), we used

Feature	Dimension	Model Size	Extraction	Scaling	Recognition
Gabor	10240	101.2 MB	488 ms	1224 ms	9593 ms
Gabor, ROI	2400	5.3 MB	160 ms	2 ms	1750 ms
Gabor, Texture, ROI	4800	9.6 MB	311 ms	10 ms	2310 ms
HoG	34596	159 MB	410 ms	1063 ms	9251 ms
HoG, ROI	3456	59 MB	188 ms	2 ms	1957 ms
HoG, Texture, ROI	12800	90.2 MB	388 ms	1100 ms	8957 ms
Texture	4096	23,2 MB	438 ms	114 ms	2204 ms
Texture, ROI	7	21,6 MB	22 ms	2 ms	3 ms
SIFT	32768	119.8 MB	1285 ms	6436 ms	11971 ms
SIFT, ROI	5862	13 MB	588 ms	2 ms	1502 ms

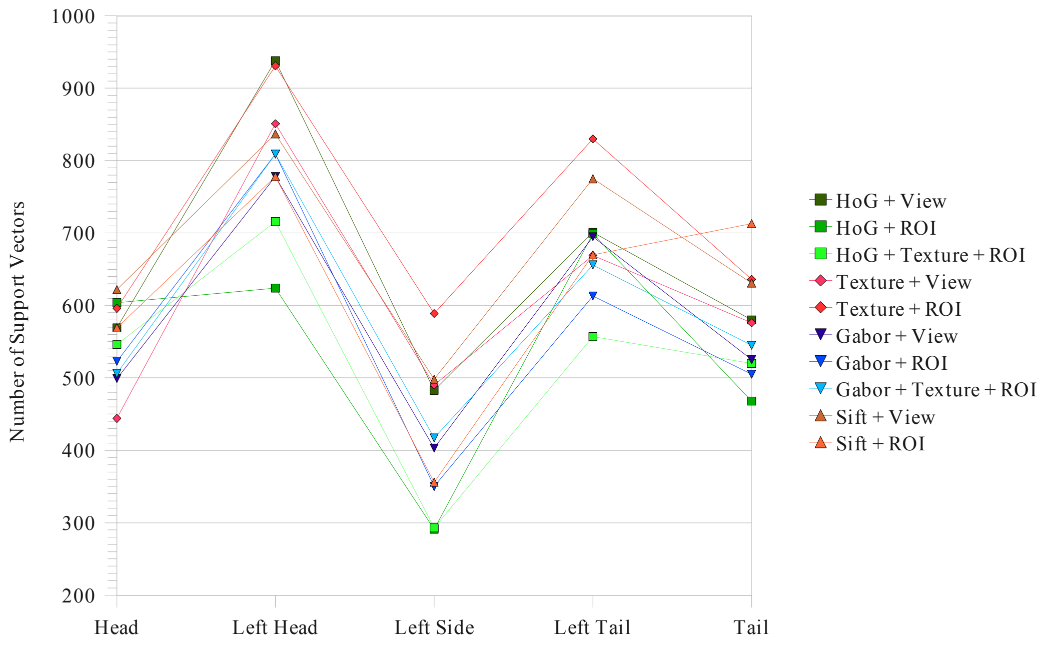
Table 4.13: This table shows the dimension of the feature vector, the size of the classification model and the consumed time for feature extraction, scaling and classification of a test image

a SVM classifier with the described training method and v-fold cross validation with $v = 5$.
We can see, that the LoG approach performs very poorly although we use a partial mask to extract only ROIs. This result might be caused by the LoG features which are very sensible to image noise like reflections or shadows. Further, we used the k-nearest neighbour classifier which does not scale the feature values and which assigns the most common label within the k nearest templates. Up to know, we use a SVM to classify data. This classifier first scales the data and transforms it to an high dimensional space in order to achieve a feature space which might be easier to separate. The technique is obvisouly more suitable for our needs. So we change the classifier and we concentrate the extraction on some more auspicious kind of features.
Slightly better results than with LoG features are possible with the texture information extracted from vehicle parts (Texture + ROI). But, the SVM needs the highest number of support vectors to separate the feature space within all methods. This indicates that the classiciation of the texture information is difficult and using only texture information is unsuitable for a proper recognition result. On the other hand, we achieve an increased accuracy in some experiments if we append the texture information to the existing feature vector. Thus we can say, that the texture information slightly improves the recognition if there are characteristic textures available. Those textures can be found at the head and tail view for example on the radiator grill or the vehicle lights. But as soon as there are parts for which the shape is more relevant than the texture, this leads to less recognition accuracy.
The features which achieve the best classification rate are Gabor features extracted from ROI (Gabor + ROI). Hence we conclude that the Gabor features are the most suitable features to describe object shapes in our experiments. The head and tail of a vehicle can be classified best by additionaly appending texture information.

We see, that high dimensional feature vectors makes the classification model unsuitable for fast recognition. Thus we introduced the SIFT descriptor which forms a low dimensional feature vector by using only scale invariant keypoints. The idea is to use orientation gradients as well, but to restrict those which are poorly located along edges and other image interferences. If we compare the results against the others, we see that the number of support vectors is less, but the recognition accuracy is worse, too. Hence it seems that the features are unsuitable because they don't describe the vehicle characteristics with a sufficient accuracy.
In almost every case, the view-dependent mask performs the worst and we get the best result if we extract only features from ROIs and if we append the texture information. Next to a better recognition rate, the number of support vectors is smaller and the classification model is suitable for a fast recognition, too.
To sum up, the vehicle types can be classified best by using Gabor features from the interesting parts of the vehicles. SIFT features are no choice, because the recognition accuracy is poor. Finally, the HoG features also form a good classification model which achieves nearby the same accuracy as Gabor features and which is also stuitable to support a fast recognition necessary for real time applications (video sequences).



(a) Classification accuracy for the corresponding feature vectors



(b) Number of support vectors for SVM training of the corresponding feature vectors

5 NVidia GPU Programming

5.1 Introduction

In regard to the system performance, it would be advantageous to run several computations in parallel. For example, the HoG scans of the vehicle detector which we describe in chapter 1.3 are independent from each other and more than one histograms could be computed at the same time. Likewise, the feature extraction of different vehicle parts is independent and can be done in parallel, too.
We will introduce NVidia's Compute Unified Device Architecture (CUDA) to control the Graphics Process Unit (GPU) which comes with the current NVidia graphics cards. We will see an example of application where we boost the HoG scans done by the vehicle detector.

5.2 Compute Unified Device Architecture

The GPU is a graphics processor where more transistors are devoted to data processing rather than data caching and flow control. Data-parallel processing maps data elements to parallel processing threads. This makes the GPU chipset suitable for compute-intensive, highly parallel computations.
Recently, NVidia released the Compute Unified Device Architecture (CUDA) for computing on the GPU. This architecture contains a compiler that provides a variation of the C programming language which gives developers access to the instruction set and memory of the GPU.
CUDA is available for the GeForce 8 Series including the Quadro and Tesla line. The following sections describe the programming model of CUDA.

5.2.1 Programming Model

The GPU is seen as a compute device which is capable of executing an high number of threads in parallel. Data-parallel and compute-intensive application parts can be isolated from the CPU (Host) and can be off-loaded onto the GPU (device). Such an isolated function is compiled to the device instruction set and the resulting program is called kernel.

Thread Batching

The threads of each kernel are organized in blocks. Threads within a block can cooperate efficiently by using a fast shared memory. To coordinate the memory access, the threads within a block can be suspended until all of them reached a specific synchronization point.
Each thread is identified by its thread id, which is based on the thread number (x, y) within the current block. Thus the thread id of a two dimensional block with size (D_x, D_y) is defined as $(x + y * D_y)$.
The number of threads per block is limited, but blocks of the same device with the same size can be batched together in so called grids. Thus the number of threads within a kernel invocation can be much larger. Further, the thread coorporation between grids is reduced to a minimum. Threads of different grids cannot communicate and cannot be synchronized with each other. Due to this restriction, the kernel can run the grids sequentially if there are only few parallel capabilities. In the other case, the grids can be executed in parallel.
Each block is identified by its block id, which is based on the block number (x, y) within the current grid. Similar to the thread identification, the block id of a two dimensional grid with size (D_x, D_y) is given as $x + y * D_x$.

Memory Model

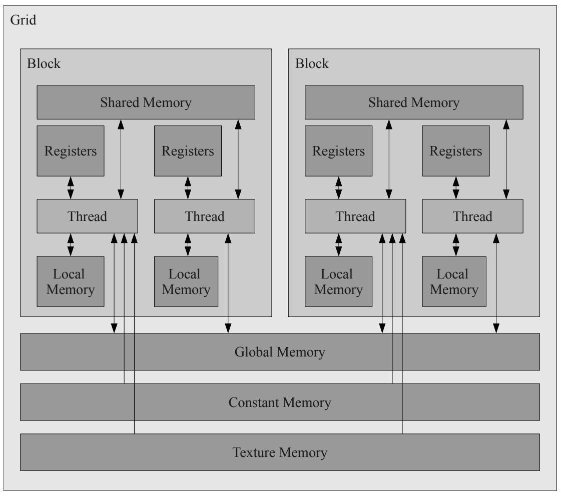


Figure 5.1: Memory spaces of various scopes for the CUDA architecture

In the CUDA architecture, the host and the device both maintain their own DRAM memory. A thread can access the device memory through a set of memory spaces of various scopes shown in figure 5.2.1. We can see, that threads within a block can communicate by using a block-wide shared memory. Additional to that, every thread has fast-accessible local memory.

Thread communication between different blocks can be done by using global, constant or texture memory. The global memory is read and writeable for every thread while the constant memory and the texture memory is read-only for the thread. By using the CUDA API, one can copy data from the host memory to the corresponding device memory.
In order to support thread batching, the communication between threads of different grids is restricted so that each grid can be executed either sequentially or in parallel.

5.3 Example of Application

As mentioned, the vehicle detector scans the input image by using a sub-window. This window is moved over the input image and for each scan, an histogram of orientated gradients is computed for the inner pixels. To achieve a proper detection rate, we need a considerable number of scans. Fortunately, the scans are independent from each other and we can run them in parallel.
The following example briefly describes the application of the CUDA extension in the HoG algorithm which was originally written by Dr. Yan Li (yanli@andrew.cmu.edu). We will see how we can change the algorithm so that we can run the compute-intensive parts in parallel.

5.3.1 Histogram of Orientated Gradients

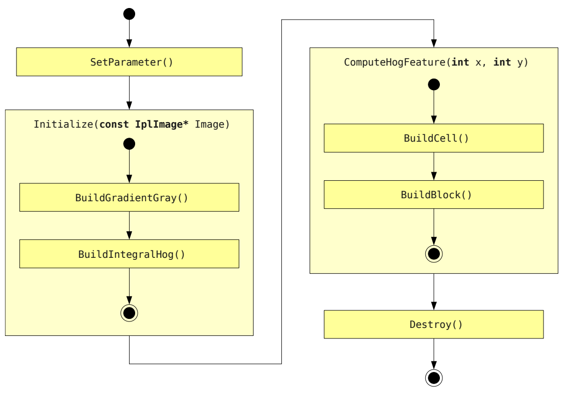


Figure 5.2: Design of the original HoG algorithm

First of all, we should have a look at the original C++ application which extracts an HoG feature vector, the so called descriptor. Figure 5.3.1 shows an activity diagram for the classical HoG extraction.
At the very beginning, *SetParameter()* is called to determine all necessary parameters like number of cells, number of blocks or descriptor length. The block

size, cell size and the number of orientations for which we want to compute the magnitudes should be given in order to determine those parameter. By default, we use a block size of 16×16 pixels, a cell size of 8×8 pixels and in total nine different orientations. The user can specify the size of a patch window for which the histogram is computed. In our experiments, the patch size corresponds to the image size and to the size of the partial mask respectively. In order to run a HoG scan, a patch size of 128×128 pixels is used.
In the next step, the *Initialize()* method is called with a pointer to the input image. In the initialization step, we compute the orientation gradients for each pixel in *BuildGradientGray()* and create an integral image in *BuildIntegralHog()* by using the magnitudes and angles of that gradients.
Finally, *ComputeHogFeature()* is called with the position of the patch window. In this method, the gradients of all pixels within one cell are summed up. We can boost this process by using the integral image which allows a very fast computation of rectangle sums as described in chapter 3.1.3. After the cell values are computed, *BuildBlock()* is called for block normalization. *ComputeHogFeature()* finally returns the descriptor as a float array which contains the magnitude for each orientation and cell.

As described in the previous sections, we can isolate data-parallel and compute-intensive program parts in order to off-load them onto the device. If we look at figure 5.3.1, we see the design of the optimized scanning algorithm. To run several scans on the same input image, we can create the integral image once and copy it to the device memory where the kernel can use the integrals to form the cell and block sums. Thus the initialization is independent from the number of scans and can be done sequentially in a pre-processing step.
The idea is to isolate *ComputeHogFeature()* and to off-load the feature computation to the device. To convert this method into a kernel, we have to change the following parts in the code:
First of all, we have to allocate device memory so that all threads of different blocks can access the integral image. After the pre-processing step, the integral image is not changed anymore and every thread needs only read-access to the allocated memory. Thus, one of the global, constant or texture memory can be used.
Further, we have to define a descriptor matrix in the global memory where each thread can store the resulting descriptors. Each line of the descriptor matrix contains the descriptor float array for the corresponding scan index.
Finally, we start the corresponding number of threads and give the scan index and pointers to the integral image and to the descriptor matrix. Each thread performs the assigned number of scans, computes the hog features by using the integral image and stores the resulting descriptor in the corresponding line of the descriptor matrix.
Another synchronization point ensures, that each thread has finished all scans before we copy the final descriptor matrix back to the host memory by using *cu-*

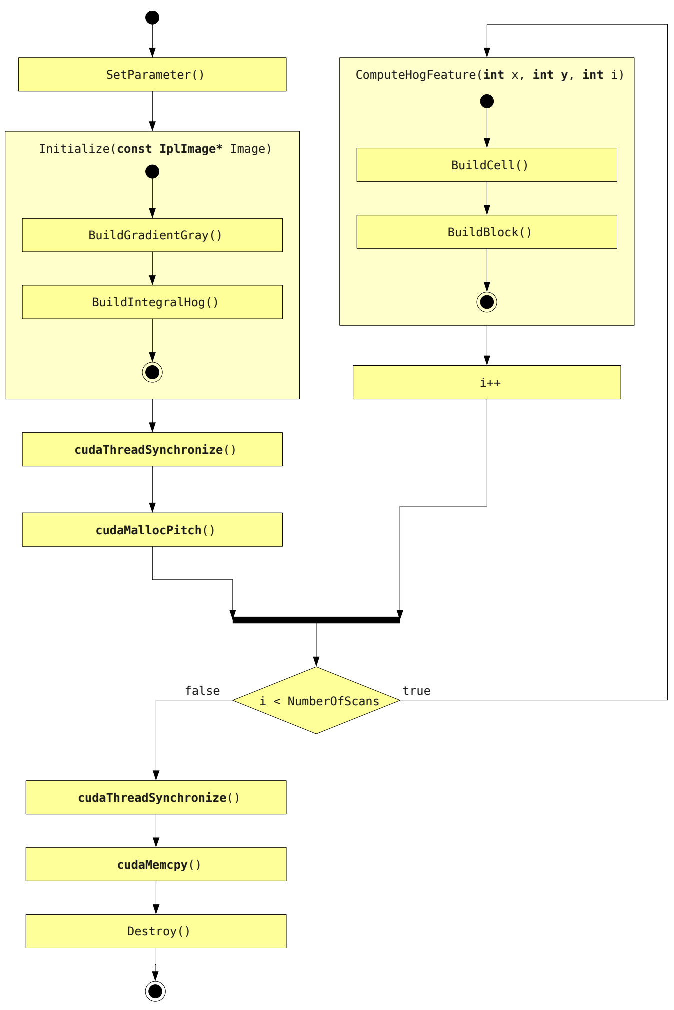


Figure 5.3: Design of the HoG scan algorithm

daMemcpy(). We can now access the descriptor matrix and can perform further algorithm to determine the vehicle position.

5.3.2 Evaluation & Conclusion

Figure 5.3.2 shows the computation time for the corresponding number of scans. If we run the HoG scans sequentially on the CPU, the overhead increases linear to the number of scans. By using the CPU, we need about 5114 ms to perform about 70000 image scans. But if we use the GPU, the scans are finished in about 599 ms. Thus by using the GPU, we can perform the scans about 8.5 times faster. The complete time statistics of our experiments can be found in appendix A.4
We run serveral experiments with a different number of scans performed by each thread. We see that one scan per thread is the best ratio. This is because CUDA is using light-weight threads and the creation of a thread needs significantly less

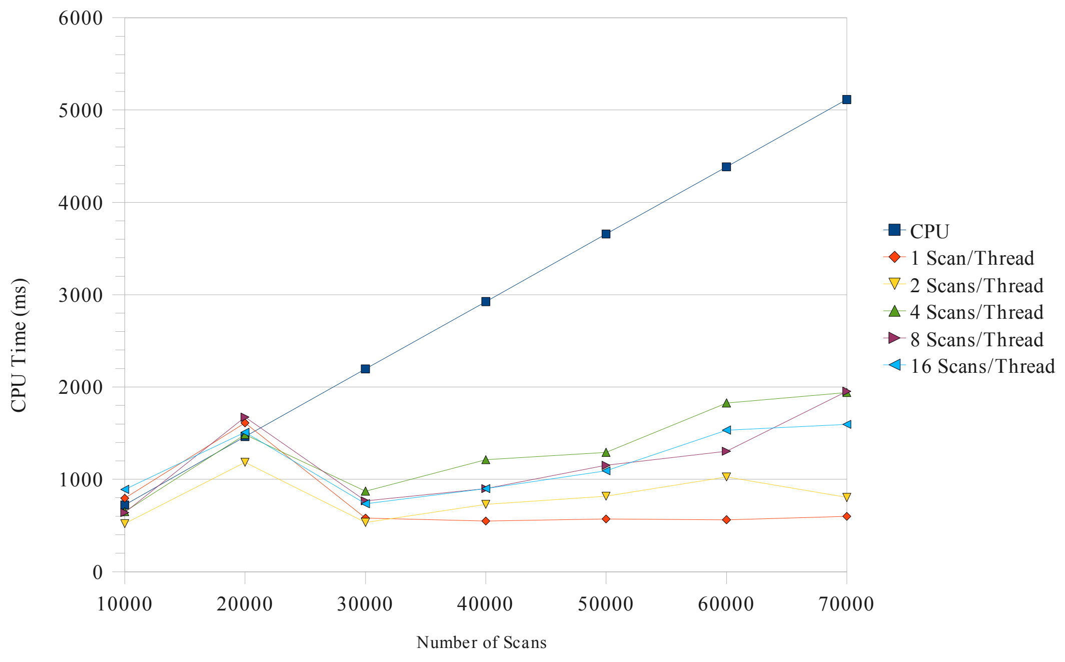


Figure 5.4: CPU and GPU times for HoG scans

overhead than computing the HoG descriptor.
To sum up, we can say that we achieve a speedup of 8.5. In comparison to the original vehicle detector, we can strongly increase the number of scans and thanks to the data parallel processing, the vehicle detector running on the GPU still performs faster than the one running on the CPU. The disadvantage is, that the GPU is only supported by modern NVidia graphics cards and thus we need special hardware in order to run the vehicle registration, detection and recognition software.

6 Future Work

6.1 Feature Extraction

The feature extraction is a fundamental part of the vehicle recognition. In our work, we used several different feature extraction methods. Some of them returned high dimensional feature vectors which formed a prohibitive huge training database, so that the training of the SVM model takes several days. If we think about application, we should keep the feature vector length as short as possible. To recognize a vehicle, we first have to extract the feature vectors and then we can classify the vectors by using the trained SVM model. If the feature vectors are too large, the feature extraction and the classification are very time consuming and the method is unsuitable for recognition in real time.
There are several approaches to reduce the feature vector dimension. The first naive method is to find a method which creates a vector with a small dimension. We saw in our experiments, that such features often lead to a weak recognition accuracy like the partial texture information or the SIFT features.
Another possibility is to use Principal Component Analysis (PCA) to reduce the dimension of the feature space. This approach tries to transform the data to a new lower dimensional coordinate system while minimizing the mean square error. Thus the data is transformed into a new structure which best explains the variance in the data. In doing so, we could achieve a lower dimensional feature space which results in smaller feature vectors and SVM models. On the other hand, we need more time for doing the transformation on the input data.

6.2 Classifier Design

Next to the feature extraction, the classification is also a fundamental part for vehicle recognition. Given a data set and a classification model, the classifier returns the most likely category for that sample. In our experiments, we use two different classifiers.
First, we tried a k-nearest neighbour classifier. The disadvantage of k-nearest neighbour was the overhead to compute the distance to all neighbours. Given an input feature vector, we have to compare this vector to all other vectors of the training database.

Other than k-nearest neighbour, the SVM creates a classification model which size depends on the number of support vectors which are necessary to separate the training data. Thus the overhead depends on the separability of the feature space. We introduced several kernels to map the feature space into an high dimensional space in order to make it more suitable to separate. In our experiments we used the recommended radial basis function as kernel 3.15. This kernel has less numerical difficulties and less hyperparameters as for example the polynomial kernel, but it would be interesting to compare the accuracy for other kernels as well.

6.3 Preprocessing

The results of our experiments show, that the recognition accuracy strongly depends on the pre-processing steps which separate the background and the foreground. In almost every approach, the view-dependent mask leads to weaker accuracy in comparison to the partial view-dependent mask which removes non-interesting parts from the background as well from the vehicle itself. Due to the detection system, we can define the position and size of the ROIs relative to the vehicle position dependent on the current view. In our experiments, we defined the rectangles shown in table 2.5(d) by hand and thus the position and size might not be optimal. Adverse chosen rectangles could significantly sophisticate the results of our experiments. This problem could be solved by determining the optimal mask parameters by using a set of images as training set.
Further, there might be other approaches which are more effective to separate the background from the foreground. In "Robust Background Subtraction with Foreground Validation for Urban Traffic Video", Sen-Ching S. Cheung and Chandrika Kamath introduce a technique to subtract the background from complex scenes where vehicles are moving at different and varying speeds. By using a Kalman filter and validating foreground pixels, the technique significantly improves the results but raises another problem with non moving vehicles which will be merged with the background.

A Appendix

A.1 Classification Rates

	Head		Left Head		Left Side		Left Tail		Tail	
Feature	nSV	Rate	nSV	Rate	nSV	Rate	nSV	Rate	nSV	Rate
LoG, ROI		72,60		71,79		68,55		50,04		76,39
HoG	569	78.71	938	76.88	483	82.58	701	82.03	580	85.20
HoG, ROI	604	79.69	624	85.79	291	94.11	699	87.12	468	88.81
HoG, Texture, ROI	546	83.30	716	**87.58**	293	95.49	557	89.83	520	91.19
Texture	444	87.01	851	79.33	490	73.93	669	78.01	576	83.88
Texture, ROI	596	72.60	931	71.93	589	68.42	830	72.60	636	76.39
Gabor	499	89.41	778	85.14	403	90.98	695	84.00	525	**91.80**
Gabor, ROI	523	85.69	809	87.01	350	**96.49**	613	**90.24**	505	91.10
Gabor, Texture, ROI	506	**90.72**	809	86.29	417	93.86	656	88.43	545	91,72
SIFT	622	81.33	837	77.17	498	83.21	775	80.48	631	85.46
SIFT, ROI	569	75.11	778	72.29	356	79.07	670	74.49	713	80.18

Table A.1: Recognition accuracy (Rate) in percent figured out by v-fold cross validation and number of support vectors (nSV) which are necessary to separate the feature space.

A.2 SVM Parameter

	Head		Left Head	
Feature	C	γ	C	γ
Hog	8	8.63167457503e-05	8	0.000410593952761
Hog, ROI	8	0.0009765625	8	0.00116133507324
Hog, Texture, ROI	8	0.0009765625	8	0.00116133507324
Texture	32	0.0001220703125	32	0.000345266983001
Texture, ROI	0.03125	0.0131390064883	0.03125	0.5
Gabor	128	1.52587890625e-05	128	0.0001220703125
Gabor, ROI	128	1.81458605195e-05	32	0.0001220703125
Gabor, Texture, ROI	32	2.15791864376e-05	8	2.56621220475e-05
SIFT	8	0.0001220703125	128	3.0517578125e-05
SIFT, ROI	8	0.001953125	32	0.00048828125

	Left Side		Left Tail	
Feature	C	γ	C	γ
Hog	32	7.25834420778e-05	128.0	0.0001220703125
Hog, ROI	32	0.00020529697638	8	0.000580667536622
Hog, Texture, ROI	32	0.00020529697638	8	0.000580667536622
Texture	512	8.63167457503e-05	128.0	3.62917210389e-05
Texture, ROI	0.03125	0.0078125	0.03125	0.0078125
Gabor	32	6.103515625e-05	32	0.00020529697638
Gabor, ROI	32	2.56621220475e-05	32	2.56621220475e-05
Gabor, Texture, ROI	8	2.15791864376e-05	8	2.56621220475e-05
SIFT	32	7.25834420778e-05	128	0.00020529697638
SIFT, ROI	128	0.00010264848819	32	0.000410593952761

	Tail	
Feature	C	γ
Hog	128.0	1.52587890625e-05
Hog, ROI	8	0.000345266983001
Hog, Texture, ROI	8	0.000345266983001
Texture	128.0	0.000172633491501
Texture, ROI	0.03125	0.5
Gabor	128.0	1.81458605195e-05
Gabor, ROI	128.0	2.15791864376e-05
Gabor, Texture, ROI	32	3.0517578125e-05
SIFT	32	3.0517578125e-05
SIFT, ROI	32	0.001953125

Table A.2: Error penality C and kernel parameter γ for the radial basis function which is used by the SVM within our experiments.

A.3 Rectangles of Interest

View	Part	X	Y	Height	Width
0	Front Lights	64	192	224	32
0	Windshield	160	96	32	96
1	Front Lights	64	176	128	32
1	Windshield	128	128	32	64
1	Side Profile	208	128	16	96
2	Windshield	112	144	48	64
2	Rear Window	224	144	48	64
2	Engine Cover	64	160	64	32
2	Back Profile	256	160	16	48
3	Back Lights	160	160	128	48
3	Back Profile	208	128	32	96
3	Side Profile	112	144	80	32
4	Back Lights	80	176	192	48
4	Back Profile	160	112	32	144
4	Side Profile	224	112	48	128

Table A.3: Pixel coordinates (X,Y) and pixelsizes for rectangles (ROI) containing the most characteristical vehicle parts. Those ROI are used in our experiments for images of 356 × 356 pixels.

A.4 CUDA Run Time Statistics

		Total Scans						
Chipset	Scans/Thread	10000	20000	30000	40000	50000	60000	70000
CPU	0	728,67	1462,57	2196,24	2924,74	3658,09	4384,07	5114,06
GPU	1	794,98	1610,54	580,7	548,55	570,03	562,77	599,21
GPU	2	519,25	1183,41	533,72	728,29	817,41	1025,32	803,72
GPU	4	651,42	1483,92	872,47	1213,6	1293	1827,13	1940,36
GPU	8	639,29	1673,24	767,93	899,13	1151,23	1304,77	1954,21
GPU	16	891,56	1509,06	736,88	899,54	1093,57	1533,2	1594,98

Table A.4: Total times in ms consumed by running the corresponding number of HoG scans on CPU and GPU, respectively.

List of Figures

List of Tables

Bibliography

[1] C. S. Cheung and C. Kamath. Robust background subtraction with foreground validation for urban traffic video. *EURASIP Journal on Applied Signal Processing*, 14:1–11, 2005.

[2] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Special invited paper. additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, 2000.

[3] M. Lades, J.C. Vorbruggen, J. Buhmann, J. Lange, C. von der Malsburg, R.P. Wurtz, and W. Konen. Distortion invariant object recognition in the dynamic link architecture. *Computers, IEEE Transactions on*, 42(3):300–311, Mar 1993.

[4] Peter Shin, Hector Jasso, Sameer Tilak, Neil Cotofana, Tony Fountain, Linjun Yan, Mike Fraser, and Ahmed Elgamal. Automatic vehicle type classification using strain gauge sensors. *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on*, pages 425–428, March 2007.

[5] T.R. Lim and A.T. Guntoro. Car recognition using gabor filter feature extraction. *Circuits and Systems, 2002. APCCAS '02. 2002 Asia-Pacific Conference on*, 2:451–455 vol.2, 2002.

[6] R. Gonzalez and R. Woods. *Digital Image Processing*. Addison-Wesley, 1992.

[7] Richard Szeliski. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis.*, 2(1):1–104, 2006.

[8] Hong bo Deng, Lian wen Jin, Li xin Zhen, and Jian cheng Huang. A new facial expression recognition method based on local gabor filter bank and pca plus lda. *International Journal of Information Technology*, 11, 2005.

[9] N. Dalai and B. Triggs. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:886–893 vol. 1, June 2005.

[10] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 1:I–511–I–518 vol.1, 2001.

[11] D.G. Lowe. Object recognition from local scale-invariant features. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 2:1150–1157 vol.2, 1999.

[12] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[13] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[14] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning).* The MIT Press, December 2001.

[15] Vladimir N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[16] C. W. Hsu, C. C. Chang, and C. J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 2003.

[17] Vadim Pisarevsky and et al. Opencv, the open computer vision library, 2008. `http://mloss.org/software/view/68/`.